第 3 章

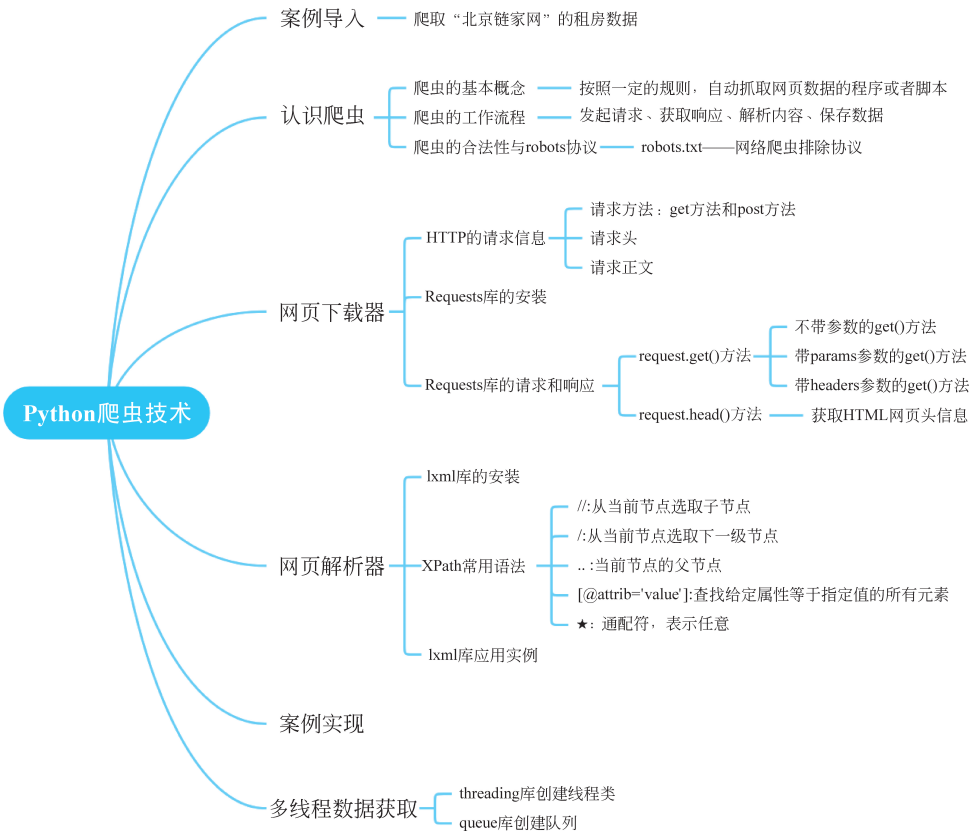
Python 爬虫技术

在大数据时代，人类社会的数据种类和规模“爆炸式”增长，数据已经渗透到每一个行业和业务领域，挖掘数据背后的知识和价值成为热门研究课题。那么数据从哪里来？可以在网络上搜索企业或政府公开的数据，但公开数据大部分针对某个领域，而且数据量较小，很难满足多样化的数据需求；可以从数据平台购买数据，这样成本比较高。因此，如果想获取多样化的、成本又比较低的数据，可以使用网络爬虫技术获取数据。本章主要介绍 Python 爬虫获取数据的基础知识和常用方法，重点讲解网页下载库 Requests 和网页解析库 lxml 及 XPath 语法。最后将爬虫技术应用于综合实战项目中爬取“北京链家网”的租房数据，爬取的数据用于后续章节的数据分析及数据挖掘。

本章学习目标

- 了解网络爬虫的基本概念。
- 了解爬虫的合法性和 robots 协议。
- 理解网络爬虫的工作原理。
- 掌握网页下载器 Requests 库的使用。
- 掌握网页解析器 lxml 库的使用。
- 掌握网络爬虫在综合实战项目中的应用。
- 掌握多线程的数据爬取。

本章思维导图



3.1　案例导入

本书综合实战项目为房屋租金数据的获取、分析与挖掘，从本章开始逐步实现该项目的任务要求。首先，该实战项目的数据来自于“北京链家网”的租房数据，网址为 https://bj.lianjia.com/zufang/。

综合实战项目数据获取任务要求如下。

从“北京链家网”爬取租房数据的主要信息，包括城区名(district)、街道名(street)、小区名(community)、楼层信息(floor)、有无电梯(lift)、面积(area)、房屋朝向(toward)、户型(model)和租金(rent)等。

3.2　认识爬虫

3.2.1　爬虫的基本概念

网络爬虫，又称为网页蜘蛛、网络机器人，是按照一定的规则，自动地抓取网页数据的程序或者脚本。浏览器的网页内容由 HTML、JS、CSS 等组成，爬虫就是通过分析 HTML 代

码,从中获取想要的文本、图片及视频等资源。

(1) 爬虫的分类。根据实现技术的不同,爬虫一般分为 4 种类型:通用爬虫、聚焦爬虫、增量式爬虫、深层爬虫。实际的网络爬虫系统通常是几种爬虫技术相结合实现的。

① 通用爬虫:又称全网爬虫,爬取目标是整个 Web,数据量庞大。这种爬虫对爬取速度和存储要求比较高,主要应用于大型搜索引擎,提供大而全的内容来满足各种不同需求的用户。

② 聚焦爬虫:又称主题爬虫,选择性爬取与预先定义好的主题相关的网页。聚焦爬虫只爬取某些主题的页面,大大节省爬虫运行时所需的网络资源和硬件资源。

③ 增量式爬虫:在爬取网页的时候,只爬取内容发生变化的网页或者新产生的网页,对于未发生内容变化的网页,则不会爬取。它在一定程度上保证所爬取的网页是尽可能新的网页。例如,电影网站会更新热门电影,增量式爬虫监测此网站的电影更新数据,抓取更新后的页面数据。增量式爬虫抓取的数据量变小了,但爬行算法的复杂度和实现难度有所提高。

④ 深层爬虫:爬取目标是互联网中深层页面的数据。网站页面分为表层页面和深层页面,表层页面是指使用超链接可以到达的静态网页为主构成的 Web 页面。深层页面是指不能通过静态链接获取的、隐藏在搜索表单后的、只有用户提交一些关键词才能获得的 Web 页面。例如,用户注册后内容才可见的网页就属于深层页面,深层爬虫就是爬取深层页面的内容。

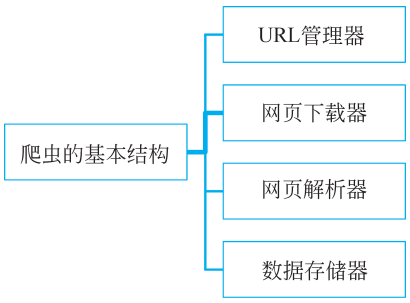


图 3-1 爬虫的基本结构图

(2) 爬虫的结构。网络爬虫的主要任务是从网页上抓取目标数据。为了完成这一任务,一个简单的爬虫主要由四个部分组成,分别是 URL 管理器、网页下载器、网页解析器、数据存储器。爬虫的基本结构如图 3-1 所示。

① URL 管理器:主要功能是存储和获取 URL 地址以及 URL 去重。包括待爬取的 URL 队列和已爬取的 URL 队列,防止重复抓取 URL 和循环抓取 URL。

爬虫从待抓取的 URL 队列中读取一个 URL,将 URL 对应的网页下载下来,将 URL 放进已抓取的 URL 队列,如果从已下载的网页中解析出其他 URL,和已抓取的 URL 进行比较去重,将不重复的 URL 放入待抓取的 URL 队列,从而进入下一个循环。

② 网页下载器:主要功能是下载网页内容。将目标 URL 地址所对应的网页下载到本地,然后将网页转换成一个字符串。实现 HTTP 请求功能常用的两个库:urllib 库和 Requests 库。urllib 库是 Python 官方基础模块,可以直接调用。Requests 库是一个第三方库,功能丰富,使用方便,本书主要使用 Requests 库。

③ 网页解析器:主要功能是解析网页内容。将一个网页字符串进行解析,提取出有用的信息。常用的网页解析器有正则表达式、lxml 库和 Beautiful Soup 库。

④ 数据存储器:主要功能是存储数据。将网页解析器提取的信息保存到文件或数据库中。

网页下载器和网页解析器是爬虫的两个核心部分，3.3 节和 3.4 节详细介绍它们的工作过程。

3.2.2　爬虫的工作流程

爬虫的工作流程如图 3-2 所示。

图 3-2　爬虫的工作流程

（1）Request：发起请求。爬虫首先通过 HTTP 库，向待爬取的 URL 地址对应的目标站点发起请求，即发送 Request，等待服务器响应。

（2）Response：获取响应。如果服务器正常工作，收到请求后，根据发送内容做出响应，然后将响应内容（response）返回给请求者。返回的内容一般是爬虫要抓取的网页内容，例如 HTML、二进制文件（视频、音频）、文档、JSON 字符串等。

（3）解析内容。爬虫利用正则表达式、页面解析库（lxml、Beautiful Soup 等）提取目标信息。

（4）保存数据。解析得到的数据保存到本地，可以以文本、音频、视频等多种形式保存。

3.2.3　爬虫的合法性与 robots 协议

如果一个网站想限制爬虫，一般有两种方式。一种方式是来源审查，即通过 User-Agent 进行限制，只响应某些浏览器或友好爬虫的访问；另一种方式是通过 robots 协议。robots 协议全称为网络爬虫排除协议（robots exclusion protocol），又称为爬虫规则，网站通过 robots 协议告知爬虫程序哪些页面或内容不能被抓取，哪些页面或内容可以被抓取。

robots 协议以文本形式存放于网站根目录下，文件名为 robots.txt（注意均为小写字母）。例如打开浏览器，在地址栏中输入 https://www.baidu.com/robots.txt，就可以看到百度 robots 协议的全部内容，下面列举一部分 robots 协议内容讲解百度设置的爬虫规则。

```
User-agent: Googlebot
Disallow: /baidu
Disallow: /s?
Disallow: /shifen/
Disallow: /homepage/
Disallow: /cpro
Disallow: /ulink?
Disallow: /link?
Disallow: /home/news/data/
Disallow: /bh

User-agent: *
Disallow: /
```

User-agent 用于描述规则对哪些爬虫有效。第一组规则中的 User-agent 是

Googlebot,这是 Google 网页抓取爬虫,也称为信息采集软件,下面列出的规则是针对 Google 爬虫的约束。一般来说,规则由 Allow 和 Disallow 开头,Allow 指定允许访问的网页,Disallow 指定不允许访问的网页。规则是以正斜线"/"开头的特定的网址或模式,例如,Disallow: /baidu 表示不允许爬虫访问网站中名为 baidu 的目录或页面;Disallow: /s? 表示不允许爬虫访问网站中名为"s?"字符串的目录或页面;Disallow: /shifen/表示不允许爬虫访问网站中名为"shifen"的目录及其中的一切;其他规则大家可以自行解释。第二组规则表示所有的爬虫都不能访问的目录("*"代表所有,"/"代表根目录)。

注意,robots 协议是国际互联网界通行的道德规范,是一个协议,而不是一个命令,不是强制执行,所以需要大家自觉遵守。使用爬虫时严格审查所抓取的内容,如发现涉密信息,应及时停止并删除,避免触碰法律底线。

在大数据时代,人们释放了许多个人隐私信息,带来了一系列问题,例如信息隐私被肆意侵害泄露等。2020 年 7 月,张某通过计算机技术手段入侵了某教育网站的数据库,获取了该网站备份的 12 万余条学生信息,其中包括学生姓名、性别、户籍、身份证号、学历等内容的有效信息 1.8 万余条,并在某论坛上出售这些学生信息。张某非法获取并利用其他公民的相关个人信息,情节尤其严重,构成了非法侵害其他公民的相关个人信息的违法犯罪行为。

从计算机专业技术人员角度来讲,对于大数据的挖掘应该遵守行为规范,加强行业自律,符合伦理道德,不要恶意挖掘用户的信息,尊重个人隐私权。

3.3 网页下载器

网页下载器的主要任务是下载网页内容,那么首先要通过 HTTP 库发起请求,常用库是 urllib 库和 Requests 库。本书主要介绍 Requests 库的使用。

3.3.1 HTTP 的请求信息

爬虫要发起 HTTP 请求,就必须了解 HTTP 的请求信息。一般来说,HTTP 的请求信息包括 3 部分:请求方法、请求头和请求正文。最常用的请求方法是 get 方法和 post 方法。get 方法的作用是请求获取指定页面内容;post 方法向指定网址提交数据,进行处理请求。打开一个网站一般使用 get 方法,如果涉及向网站提交数据,例如登录,就用到了 post 方法。请求头包含客户端的一些信息,例如 User Agent 和 Cookie 等,这些信息经常用于爬虫程序中。

下面打开一个网页,查看浏览器向网站发送的 HTTP 请求。打开 Firefox 浏览器或其他浏览器,在网页空白处右击,在弹出的快捷菜单中选择"检查",浏览器下方出现子窗口,选择此窗口左侧的"网络",打开网络监视器。然后在浏览器的地址栏中输入 https://www.baidu.com,可以看到网络监视器中出现很多请求,单击最上面一个请求,右侧出现该请求的详细信息,如图 3-3 所示。这就是当前浏览器向百度服务器发起的 HTTP 请求所包含的信息。

从图 3-3 可以看到,这次 HTTP 请求所用的方法是 get 方法,请求地址为 https://www.baidu.com,状态码为 200,表明请求已经成功。请求头在消息头窗口的下方,其中这

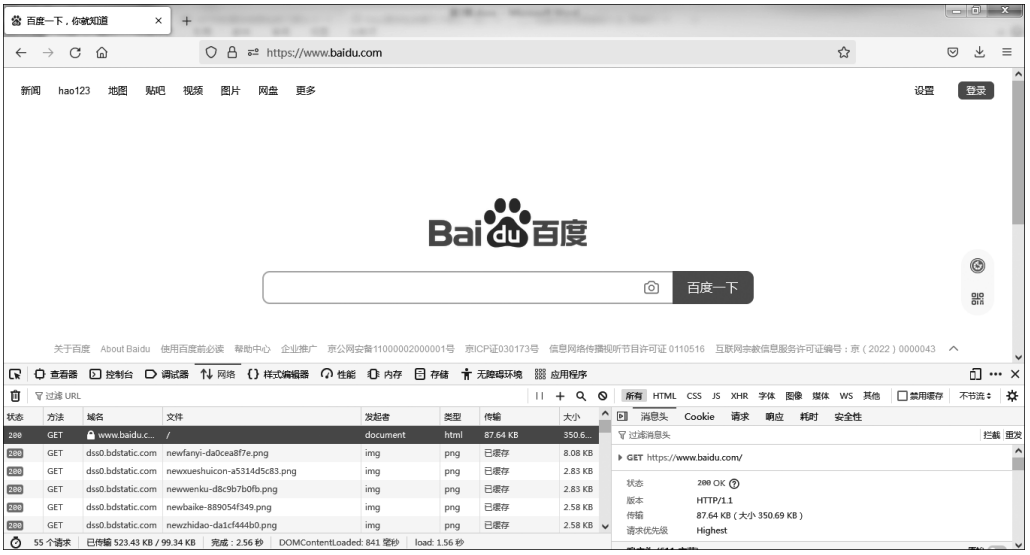


图 3-3　Firefox 浏览器的检查工具

次请求的 User Agent 信息如下：

```
Mozilla/5.0 (Windows NT 6.1; Win64; x64; rv:91.0) Gecko/20100101 Firefox/91.0
```

这个 User Agent 信息表示这次请求的发起者是 Firefox 浏览器，同时显示了浏览器的版本信息。下面列举几个常见的 User Agent 信息。

（1）Mozilla/5.0（Windows NT 6.1；Win64；x64；rv：91.0）Gecko/20100101 Firefox/91.0。

浏览器名称：Firefox。

浏览器版本号：91.0。

操作系统：Windows。

（2）Mozilla/5.0(Windows NT 6.3；WoW64；Trident/7.0；rv:11.0)like Gecko。

浏览器名称：IE。

浏览器版本号：11。

操作系统：Windows。

（3）Mozilla/5.0(Windows NT 10.0；Win64；x64) appleWebkit/537.36(khtml，like Gecko)chrome/51.0.2704.79 safari/537.36 edge/14.14393。

浏览器名称：Edge。

浏览器版本号：14.14393。

操作系统：Windows。

（4）Mozilla/5.0(compatible；Konqueror/3.5；Linux)KHTML/3.5.5(like Gecko)(Kubuntu)。

浏览器名称：Konqueror。

浏览器版本号：3.5。

操作系统：Kubuntu。

(5) Mozilla/5.0 (X11; Linux i686) AppleWebKit/535.7 (KHTML, like Gecko) Ubuntu/11.04 Chromium/16.0.912.77 Chrome/16.0.912.77 Safari/535.7。

浏览器名称：Chromium。

浏览器版本号：16.0.912.77。

操作系统：Ubuntu。

本节介绍了 HTTP 请求信息的基本内容，以及在浏览器中检查请求信息的方法，编写爬虫程序时会经常用到这些内容。下面以 Requests 库为例讲解 HTTP 请求的实现。

3.3.2 Requests 库的安装

Requests 库是公认的爬取页面最好的第三方库。它的语法非常简洁，有时可以用一条语句从网页上获取信息。Requests 库使用 pip 安装，命令如下。

```
pip install requests
```

如果已经安装了 Anaconda，Requests 不需要另行安装，Anaconda 默认安装时包含这个库。

安装好 Requests 库后，在 Python 开发环境中导入。命令如下。

```
import requests
```


扫一扫

3.3.3 Requests 库的请求和响应

Requests 库的常用方法如表 3-1 所示。如果想获取某网址的资源，可以使用 get()和 head()方法：get()方法获得该网址下的全部资源，head()方法获得该网址页面的头部资源。如果要将资源放到该网址对应的位置上，可以使用 post()、put()或 patch()方法；如果想删除该网址对应的资源，可以使用 delete()方法。这里只介绍本书中用到的 get()和 head()方法，其他方法读者可查阅相关资料。

表 3-1 Requests 库的常用方法

方　法	说　明
requests.request()	构造一个请求，各方法的基础方法
requests.get()	请求获取目标网页资源
requests.head()	请求获取目标网页资源的头部信息
requests.post()	向目标网页提交 POST 请求
requests.put()	请求向目标网址存储资源
requests.patch()	请求在目标网址进行局部修改
requests.delete()	请求在目标网址删除资源

(1) get()方法。Requests 库中 get()方法的语法格式如下。

```
requests.get(url,params=None,**kwargs)
```

其中，参数 url 表示目标网页的 URL 链接，params 是以字典或字节流的格式作为参数增加到 URL 中，kwargs 是 12 个控制访问参数，以键值对的形式表示，比较常用的关键字参数是 headers 参数。

① 不带参数的 get()方法。代码如下。

```
r=requests.get("https://www.baidu.com")
print(r.status_code)
```

输出结果为：

```
200
```

上面的代码使用 get()方法获取百度首页，返回了一个包含百度首页内容的响应对象(Response 对象)，并赋值给 r。

响应对象的属性包括获取本次请求的响应状态、响应头和响应体等信息。例如，在上面的代码中，*r*.status_code 表示响应状态，响应状态码是 200，表示响应成功，404 为找不到页面，502 为服务器错误等。响应对象常用的属性如表 3-2 所示。

表 3-2　响应对象常用的属性

属　性	说　明	属　性	说　明
status_code	HTTP 请求的响应状态	header	响应头信息
text	响应内容的字符串形式	content	响应内容的二进制形式
encoding	响应内容的编码方式以及修改编码		

响应对象的 text 属性可以输出页面内容，encoding 属性可以查看或修改页面的编码形式，例如：

```
r=requests.get("https://www.baidu.com")
print(r.encoding)
print(r.text)
```

输出结果如图 3-4 的(a)图所示，Requests 库使用 ISO-8859-1 解码百度首页的内容，但其中有一些乱码，ISO-8859-1 不能解码其中的某些内容，修改 encoding 属性，使 *r*.text 正常输出网页内容。代码如下。

```
r.encoding='UTF-8'
print(r.encoding)
print(r.text)
```

输出结果如图 3-4 的(b)图所示，这时百度首页内容能够正常输出了，对比两次 *r*.text 的输出结果，出现乱码的原因是中文在 ISO-8859-1 编码方式下不能正常编码。

```
ISO-8859-1
<!DOCTYPE html>
<!--STATUS OK--><html> <head><meta http-equiv=content-type content=text/html;charset=utf-8><meta http-equiv=X-UA-
Compatible content=IE=Edge><meta content=always name=referrer><link rel=stylesheet type=text/css href=https://
ss1.bdstatic.com/5eN1bjq8AAUYm2zgoY3K/r/www/cache/bdorz/baidu.min.css><title>ç™¾åº¦ä¸ä¸ï¼ä½ å°±ç¥é“</title></
head> <body link=#0000cc> <div id=wrapper> <div id=head> <div class=head_wrapper> <div class=s_form> <div
class=s_form_wrapper> <div id=lg> <img hidefocus=true src=//www.baidu.com/img/bd_logo1.png width=270 height=129>
</div> <form id=form name=f action=//www.baidu.com/s class=fm> <input type=hidden name=bdorz_come value=1> <input
type=hidden name=ie value=utf-8> <input type=hidden name=f value=8> <input type=hidden name=rsv_bp value=1>
<input type=hidden name=rsv_idx value=1> <input type=hidden name=tn value=baidu><span class="bg s_ipt_wr"><input
```

(a) ISO-8859-1编码下的网页输出结果

```
UTF-8
<!DOCTYPE html>
<!--STATUS OK--><html> <head><meta http-equiv=content-type content=text/html;charset=utf-8><meta http-equiv=X-UA-
Compatible content=IE=Edge><meta content=always name=referrer><link rel=stylesheet type=text/css href=https://
ss1.bdstatic.com/5eN1bjq8AAUYm2zgoY3K/r/www/cache/bdorz/baidu.min.css><title>百度一下，你就知道</title></head>
<body link=#0000cc> <div id=wrapper> <div id=head> <div class=head_wrapper> <div class=s_form> <div
class=s_form_wrapper> <div id=lg> <img hidefocus=true src=//www.baidu.com/img/bd_logo1.png width=270 height=129>
</div> <form id=form name=f action=//www.baidu.com/s class=fm> <input type=hidden name=bdorz_come value=1> <input
type=hidden name=ie value=utf-8> <input type=hidden name=f value=8> <input type=hidden name=rsv_bp value=1>
<input type=hidden name=rsv_idx value=1> <input type=hidden name=tn value=baidu><span class="bg s_ipt_wr"><input
```

(b) UTF-8编码下的网页输出结果

图 3-4　Response 对象的 text 属性输出页面内容

② 带 params 参数的 get()方法。

如果在百度首页搜索“Python”，搜索结果的 URL 链接为：

```
https://www.baidu.com/s?ie=UTF-8&wd=Python
```

这个 URL 中“?”后面有两个参数“ie”和“wd”，分别表示编码形式和搜索关键字。Requests 使用 params 为 URL 提供这样的参数。params 必须是字典形式，例如在百度首页搜索“Python”的 URL 页面的 get()方法编写如下。

```
mypara={'ie':'UTF-8', 'wd':'Python'}
r=requests.get("https://www.baidu.com/s", params=mypara)
print(r.url)
```

输出结果为：

```
https://www.baidu.com/s?ie=UTF-8&wd=Python
```

③ 带 headers 参数的 get()方法。

每一个 HTTP 请求都包含一个请求头 headers，在图 3-3 中打开百度首页时，本次请求的头部信息 headers 中记录了请求发起者是 Firefox 浏览器。很多网站不想让爬虫爬取网站内容，消耗服务器资源，所以设计了反对爬虫策略，最常用的反爬策略就是服务器通过读取 headers 中的 User Agent 值来判断访问请求来自于浏览器或其他地址。人们可以使用 headers 参数自定义请求头信息。headers 参数是一个字典形式，示例如下。

```
r=requests.get("https://www.baidu.com")          #不带 headers 参数的 get()方法
print(r.request.headers)                         #查看请求头信息
myheader={'User-Agent': 'Mozilla/5.0 (Windows NT 6.1; Win64; x64; rv:91.0)
Gecko/20100101 Firefox/91.0'}                    #User Agent 设置为 Firefox 浏览器
r=requests.get("https://www.baidu.com", headers=myheader)
                                                 #带 headers 参数的 get()方法
print(r.request.headers)
```

输出结果为：

```
{'User-Agent': 'python-requests/2.23.0', 'Accept-Encoding': 'gzip, deflate',
'Accept': '*/*', 'Connection': 'keep-alive'}
{'User-Agent': 'Mozilla/5.0 (Windows NT 6.1; Win64; x64; rv:91.0) Gecko/20100101
Firefox/91.0', 'Accept-Encoding': 'gzip, deflate', 'Accept': '*/*',
'Connection': 'keep-alive'}
```

可以看出，如果没有修改 headers 中 User Agent 值，User Agent 值是 python-requests，如果传递了 headers 参数到 Requests 请求中，那么请求的头信息被修改为相应值。

(2) head()方法。head()方法主要用于获取 HTML 网页头信息。例如，抓取百度首页的头部信息，示例代码如下。

```
r=requests.head('https://www.baidu.com')
print(r.headers)
```

输出结果为：

```
{'Cache-Control': 'private, no-cache, no-store, proxy-revalidate, no-
transform', 'Connection': 'keep-alive', 'Content-Encoding': 'gzip', 'Content-
Type': 'text/html', 'Date': 'Mon, 19 Sep 2022 09:37:04 GMT', 'Last-Modified':
'Mon, 13 Jun 2016 02:50:01 GMT', 'Pragma': 'no-cache', 'Server': 'bfe/1.0.8.18'}
```

对象 r 是一个响应对象，r.headers 获取响应头信息，也就是拟抓取网页的头信息。返回结果是一个字典，其中包括网页的缓存信息、连接方式、内容编码、内容格式、访问日期等信息。

3.4　网页解析器

从目标网址下载了页面内容后，要从 HTML 源码中提取数据，这就需要对网页进行解析。常用的网页解析工具有正则表达式、lxml 库和 Beautiful Soup 库。正则表达式采用字符串匹配的方式查找目标内容，解析速度快，但语法复杂，如果对解析速度要求比较高的爬虫，可以使用正则表达式。lxml 库支持 HTML 和 XML 的解析，使用 XPath(XML Path Language)，语法简单、解析效率高，推荐新手入门使用。Beautiful Soup 采用 Python 自带的 html.parse 作为解析器，也可以采用 lxml 作为解析器，语法比较简单，但解析速度慢。本节主要讲解 lxml 库以及 XPath 语法。

3.4.1　lxml 库的安装

lxml 不是 Python 的标准库，需要另行安装。使用 pip 安装命令如下。

```
pip install lxml
```

或者用 wheel 文件安装，首先从 http://www.lfd.uci.edu/~gohlke/pythonlibs/#lxml 中下载对应系统版本的 wheel 文件，然后运行如下安装命令。

```
pip install lxml-4.2.1-cp36-cp36m-win_amd64.whl
```

如果已经安装了 Anaconda,lxml 不需要另行安装,Anaconda 默认安装时包含这个库。


扫一扫

3.4.2 XPath 常用语法

XPath 是 XML 的路径语言,可以在 XML 和 HTML 文档中查找内容。XPath 采用了树状结构,提供了非常简明的路径选择表达式。lxml 支持 XPath 语言解析 HTML 文档中的内容,它的大部分功能包含在 lxml.etree 下,所以使用 lxml 时要从 lxml 导入 etree。

XPath 常用语法如表 3-3 所示。

表 3-3 XPath 常用语法

表 达 式	说 明
//	从当前节点选取子孙节点
/	从当前节点选取下一级子节点
..	当前节点的父节点
[@attrib='value']	查找给定属性等于指定值的所有元素
*	通配符,表示任意

下面以一段 HTML 源码为例讲解 XPath 查找并提取文本的语法。

```
from lxml import etree
html_text='''
<body>
<div id='texts'>
    <li class="css1"><a href="text1.html">First text</a></li>
    <li class="css2"><a href="text2.html">Second text </a></li>
    <li class="css3"><a href="text3.html">Third text </a></li>
  </div>
</body>
'''
```

HTML 语言中<body>和</body>之间的内容是网页主体,可以从其中提取所需要的信息。例如提取“First text”这段文字,首先给定它在该页面的路径,XPath 就是通过该路径找到文字“First text”,具体代码如下。

(1) 首先将这段源码送入 etree 的 HTML 方法中。代码如下。

```
myelement=etree.HTML(html_text)
print(type(myelement))
```

输出结果为:

```
<class 'lxml.etree._Element'>
```

得到的 myelement 实际上是一个 Element 对象,如果想看到解析的内容,可以使用如下代码。

```
print(etree.tostring(myelement).decode('utf-8'))
```

输出结果为：

```
<class 'lxml.etree._Element'>
<html><body>
<div id="texts">
    <li class="css1"><a href="text1.html">First text</a></li>
    <li class="css2"><a href="text2.html">Second text </a></li>
    <li class="css3"><a href="text3.html">Third text </a></li>
  </div>
</body>
</html>
```

(2) 查找“li”标签。Element 对象 myelement 的 XPath 方法可以查找指定路径，“//”表示从根节点开始查找。

```
li_text=myelement.xpath('//div/li')
print(li_text)
```

输出结果为：

```
<Element li at 0x791c648>, <Element li at 0x950bcc8>, <Element li at 0x950b448>]
```

运行结果是包含 3 个 Element 元素的列表，这是因为 HTML 源码中有 3 个“li”标签。要查找的内容“First text”在第一个“li”标签下，可以使用 li[1]表示，注意 XPath 语法中序号从 1 开始。

(3) 提取文本内容。HTML 源码中“First text”是第一个“li”标签下“a”标签里的文本内容，可以使用 text()获取标签的文本内容。

```
text=myelement.xpath('//div/li[1]/a/text()')
print(text)
```

输出结果为：

```
['First text']
```

输出结果是包含查找文本内容的列表，使用 text[0]即可得到“First text”字符串。

(4) 提取属性值。如果要提取“First text”对应的链接地址“text1.html”，因为链接地址“text1.html”是“a”标签 href 属性的值，可以使用@href 属性获取该链接地址。

```
link_text=myelement.xpath('//div/li[1]/a/@ href')
print(link_text)
```

输出结果为：

```
['text1.html']
```

另外，使用[@attrib='value']可以查找给定属性等于指定值的所有元素，例如第一个“li”标签的路径可以用 li[@class="css1"]，功能等价于 li[1]。使用属性值查找标签比用序

号查找更方便些,只要知道标签的属性值,而不用查看这是第几个标签。使用属性值查找方式提取标签文本内容“First text”和链接地址“text1.html”的代码如下。

```
text=myelement.xpath('//div/li[@class="css1"]/a/text()')[0]
print(text)
link_text=myelement.xpath('//div/li[@class="css1"]/a/@href')[0]
print(link_text)
```

输出结果为:

```
First text
text1.html
```

扫一扫

3.4.3 lxml 库应用实例

下面以百度首页为例介绍使用 lxml 库对真实网页源码的解析过程。假设要提取百度首页中的“新闻”一词和它对应的链接地址,lxml 库的使用过程如下。

首先需要了解“新闻”所在的路径。使用 Firefox 浏览器打开百度首页,在页面空白处右击,在弹出的快捷菜单中选择“检查”功能,打开检查窗口,最左端有一个箭头形状的按钮,使用它可以选取页面上的元素,单击箭头按钮,在百度首页上单击“新闻”,这时“查看器”子窗口中就会显示“新闻”在 HTML 源码中的位置,如图 3-5 所示。要想得到“新闻”一词的路径,浏览器提供了一种简便方法:在“查看器”子窗口中突出显示的“新闻”的 HTML 代码上右击,在弹出的快捷菜单中选择“复制”→“XPath”,即可得到“新闻”所对应的 XPath 语法的路径:/html/body/div[1]/div[1]/div[3]/a[1]。这里使用了“div”标签的序号形式来表示路径,即“新闻”一词在 div[1]→div[1]→div[3]目录下。

图 3-5 百度首页的 HTML 源码突出显示“新闻”一词

使用 Firefox 浏览器得到的 XPath 语法路径后，提取百度首页上“新闻”一词和它对应的链接地址代码如下。

```
myheader={'User-Agent': 'Mozilla/5.0 (Windows NT 6.1; Win64; x64; rv:91.0)
Gecko/20100101 Firefox/91.0'}
r=requests.get("https://www.baidu.com", headers=myheader)
myelement=etree.HTML(r.text)
news_link=myelement.xpath('//html/body/div[1]/div[1]/div[3]/a[1]/@href')[0]
news_text=myelement.xpath('//html/body/div[1]/div[1]/div[3]/a[1]/text()')[0]
print(news_link)
print(news_text)
```

输出结果为：

```
http://news.baidu.com
新闻
```

在上面的代码中，XPath 语法的路径使用了“div”标签的序号形式，也可以使用“div”标签的 id 属性值为“s-top-left”查找“新闻”一词所在的路径，例如语句 news_text = myelement.xpath('//div[@id="s-top-left"]/a[1]/text()')[0]的返回值为“新闻”。

值得注意的是，在上面的代码中，发起 Requests 请求时传递了 headers 参数，将 User-Agent 定义为 Firefox 浏览器。如果不设置 User-Agent，“新闻”所在的“div”标签的 id 属性值为“u1”，这时代码需要修改为：

```
r=requests.get("https://www.baidu.com")
r.encoding='UTF-8'          #解析内容能够正常显示汉字
myelement=etree.HTML(r.text)
news_link=myelement.xpath('//div[@id="u1"]/a[1]/@href')[0]
news_text=myelement.xpath('//div[@id="u1"]/a[1]/text()')[0]
print(news_link)
print(news_text)
```

3.5　案例实现


扫一扫

下面以 1.5 节中的综合实战项目为例，从“北京链家网”爬取包括城区名(district)、街道名(street)、小区名(community)、楼层信息(floor)、有无电梯(lift)、面积(area)、房屋朝向(toward)、户型(model)和租金(rent)等租房信息。

爬取数据的程序流程图如图 3-6 所示。

(1) 导入库。代码如下。首先导入爬取过程中所需的库，这里仅对库进行简单说明，后面会详细讲解。

```
import csv
import random
import time
import requests
import pandas as pd
from lxml import etree
```

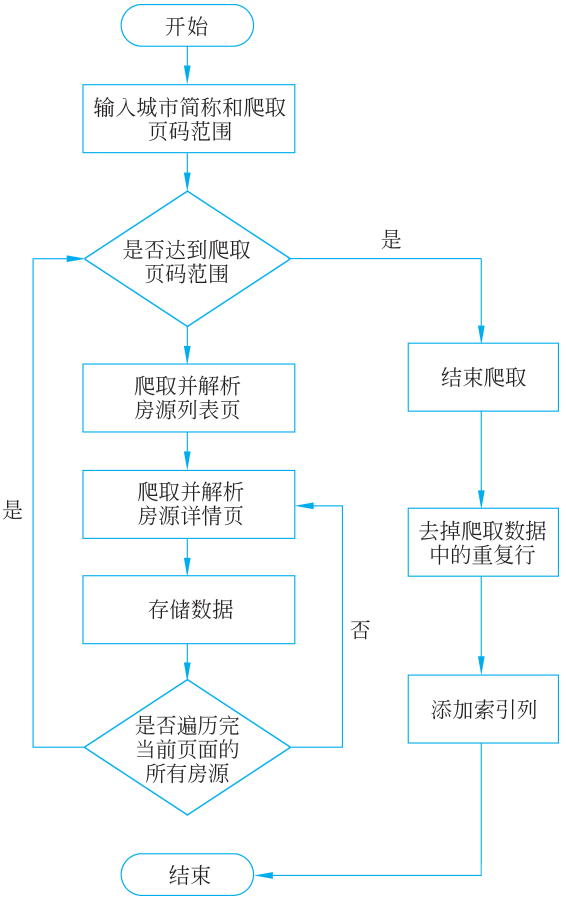


图 3-6 “北京链家网”租房数据爬取流程图

其中,requests 库用于请求指定页面并获取响应,etree 库对返回的页面进行 XPath 解析,以获取指定的数据,random 库和 time 库用于爬取过程中的一些设置,Pandas 库和 csv 库用于处理和保存文件。

(2) 输入“北京链家网”的城市简称和要爬取的房源页面的页码范围。打开租房页面,首先展现的是房源列表页,通过观察不同城市的链家网租房页面的 URL 链接可知,一个房源列表页的 URL 链接主要分为 3 部分:城市的拼音简写、页码和其他相同部分,例如下面给出三个房源列表页的 URL。

```
#北京
https://bj.lianjia.com/zufang/pg4/#contentList
#重庆
https://cq.lianjia.com/zufang/pg5/#contentList
#上海
https://sh.lianjia.com/zufang/pg6/#contentList
```

其中,第一条 URL 中的“bj”表示城市北京的拼音简写,“pg4”表示第 4 页。通过设置房源列表页 URL 的城市拼音简写,除了可以爬取北京的租房数据外,也可以爬取其他城市的

租房数据。

本案例使用城市(如 bj)和页码(pg4)的拼音简称构造一条房源列表页的 URL 链接,其中“#contentList”用于页内定位,可以忽略。


扫一扫

(3) 爬取并解析房源列表页。由于本案例拟爬取的房源信息在房源列表页和每一个房源的详情页均有分布,因此首先需要获取每一个房源详情页的 URL 链接。另外,房源列表页中还包含每一个房源所在的地理位置(所在城区、街道和小区)。下面介绍如何对房源列表页的页面信息进行分析,以获得房源详情页的 URL 链接和每一个房源的地理位置。

使用火狐浏览器打开“北京链家网”的租房页面 https://bj.lianjia.com/zufang/,在页面空白位置右击,在弹出的快捷菜单中选择“检查”功能,单击页面最左侧小箭头形状的按钮,随后指向其中的一个房源信息,在“查看器”子窗口中右击蓝色文字的 MTML 代码,选择“复制”→“整体 HTML”,得到如下的 HTML 内容(为简化分析难度,仅保留部分代码)。

```
<div class="content__list">
    <div class="content__list--item">
        <div class="content__list--item--main">
            <p class="content__list--item--title">
                <a class="twoline" target="_blank"
href="/zufang/BJ2840486736310837248.html">
                    整租·长阳国际城二区 3室1厅南/北</a>
            </p>
            <p class="content__list--item--des">
                <a target="_blank" href="/zufang/fangshan/">房山</a>-<a
href="/zufang/changyang1/" target="_blank">长阳</a>-<a
                    title="长阳国际城二区" href="/zufang/c1111053458322/"
target="_blank">长阳国际城二区</a>
                <i>/</i>
                89.00m²
                <i>/</i>南北<i>/</i>
                3室1厅1卫<span class="hide">
            <i>/</i>
            中楼层          (20层)
                </span>
            </p>
        </div>
    </div>
</div>
```

通过观察会发现,房源详情页的 URL 链接包含在 标签中的<a class="twoline">标签的 href 属性中,房源的地理位置(城区名、街道名和小区名)包含在<p class="content__list--item--des">标签中的 3 个“a”标签内。

获得房源的 URL 链接和地理位置的具体过程如下。

① 首先,通过 XPath 获取房源的 URL 链接,路径为//a[@class="content__ list--item--aside"]/@href。使用该路径可以获取当前页面中所有 class 属性为“content__list--item--aside”的“a”标签的 href 属性值,得到的结果为当前页面所有房源的 URL 链接列表 detailsUrl。

② 然后,通过 XPath 获取房源的地理位置(城区名、街道名和小区名),路径为 //p[@class="content__list--item--des"]/a/text()。使用该路径可以获取当前页面中所有 class 属性为“content__list--item--des”的“p”标签下“a”标签的文本内容,得到的结果为当前页面所有房源的地理位置列表 location。需要注意的是,每一个“p”标签下共有 3 个“a”标签,分别对应房源的城区名、街道名和小区名。

③ 最后,通过遍历列表 detailsUrl 和列表 location,将房源详情页 URL 链接和对应的城区名、街道名和小区名存放在字典中。

该部分代码如下。

```
def getPageLines(city, page):
    """
    获取指定城市和页码所在页面中的房源 URL 链接、所在地理位置(district street community),并分别存入 house 字典中
    :param city:城市简称
    :param page:要爬取的页码
    :return:字典列表
    """
    #构造房源列表页的 URL 链接
    URL="https://" + city + ".lianjia.com/zufang/pg"+ str(page)
    #构造房源详情页 URL 链接的公共部分
    baseUrl=URL.split("/")[0] + "//" + URL.split("/")[2]
    #爬取房源列表页,并处理响应信息
    response=requests.get(url=URL)
    #获取页面 HTML,并对其进行解析
    html=response.text
    myelement=etree.HTML(html)
    #提取本页面所有房源的 URL 链接和地理位置
    detailsUrl=myelement.xpath('//a[@class="content__list--item--aside"]/@href')
    location=myelement.xpath('//p[@class="content__list--item--des"]/a/text()')
    #将数据存入字典列表中
    houses=list()
    for i in range(len(detailsUrl)):
        #获取房源详情页的 URL 链接
        detailsLink=baseUrl + detailsUrl[i]
        #获取房源所在地理位置
        lineIndex=i * 3
        district=location[lineIndex]
        street=location[lineIndex + 1]
        community=location[lineIndex + 2]
        #将房源详情页 URL 链接和所在地理位置存入字典
        house={}
        house["detailsLink"]=detailsLink
        house["district"]=district
        house["street"]=street
        house["community"]=community
        houses.append(house)
    return houses
```


扫一扫

（4）爬取并解析房源详情页。从房源详情页中可以获取房屋楼层、电梯、面积、朝向、户型和租金等信息。在浏览器中打开一个房源详情页，通过火狐浏览器的“检查”功能获取房源详情页面的部分 HTML 源码。房源详情页的部分 HTML 源码如下。

```
<div class="content__aside--title">
    <span>5500</span>元/月
    (季付价)
    <div class="operate-box">……</div>
</div>
<ul class="content__aside__list">
        <li><span class="label">租赁方式：</span>整租</li>
        <li><span class="label">房屋类型：</span>3 室 1 厅 1 卫 89.00㎡ 精装修</li>
        <li class="floor">
            <span class="label">朝向楼层：</span>
            <span class="">南/北中楼层/20 层</span>
        </li>
        <li>
            <span class="label">风险提示：</span>
            <a href="https://m.lianjia.com/text/disclaimer">用户风险提示</a>
        </li>
        </ul>
    <div class="content__article__info" id="info">
            <h3 id="info">房屋信息</h3>
            <ul>
                <li class="fl oneline">基本信息</li>
                <li class="fl oneline">面积：89.00㎡</li>
                <li class="fl oneline">朝向：南北</li>
                <li class="fl oneline"> </li>
                <li class="fl oneline">维护：7 天前</li>
                <li class="fl oneline">入住：随时入住</li>
                <li class="fl oneline"> </li>
                <li class="fl oneline">楼层：中楼层/20 层</li>
                <li class="fl oneline">电梯：有</li>
        …
          </ul>
                …
</div>
```

通过分析该段 HTML 源码可知，房屋租金位于 class 属性为“content__aside--title”的＜div＞标签下的＜span＞标签中，房屋户型位于 class 属性为“content__aside__list”的＜div＞标签下的第二个“li”标签中，楼层、电梯、面积和朝向 4 个房屋信息都存在于 class 属性为“content__article__info”的＜div＞标签下的“ul”标签下的“li”标签中。通过 XPath 获取“li”标签中的内容，并对文本内容进行数据清洗，以得到字典类型的目标数据，最后将其存入对应的 house 字典。

获得房源的楼层、电梯、面积、朝向、户型和租金等信息的具体过程如下。

① 在上一步得到的 house 字典中读取房源详情页的 URL 链接，发送请求获取房源详情页面的 HTML 源码，并对其进行解析。

② 通过 XPath 获取房屋租金 rent 和房屋户型 model，得到的结果存入 house 字典。

③ 通过 XPath 获取房屋楼层、电梯、面积和朝向。因为 HTML 内容中解析得到的数据

含有空格,使用 dataCleaning()方法对其进行数据清洗,该方法同时对数据进行类型转换,最终将得到的房屋楼层、电梯、面积和朝向数据存入 house 字典。

至此,所需的数据都保存在 house 字典中,该部分代码如下。

```
def getDetail(house):
    """
    爬取并解析房源详情页的数据,将其存入字典中。
    :param house: 含有房源详情页 URL 链接和地理位置的字典
    :return: 含有案例拟爬取数据的字典
     """
    #读取房源详情页的 URL 链接
    url=house["detailsLink"]
    try:
        response=requests.get(url=url, timeout=10)
    except:
        return None,None
    #获取房源详情页面 HTML,并对其进行解析
    myelement=etree.HTML(response.text)
    #获取房屋租金和房屋户型
    house["rent"]=myelement.xpath('//div[@class="content__aside--title"]/
span/text()')[0]
    house["model"]=(myelement.xpath('//ul[@class="content__aside__list"]/li
[2]/text()'))[0].split(" ")[0]
    #获取房屋其他信息:楼层, 电梯, 面积, 朝向
    details=myelement.xpath('//div[@class="content__article__info"]/ul[1]/li/
text()')
    details=dataCleaning(details)
    house["floor"]=details["楼层"]
    house["lift"]=details["电梯"]
    house["area"]=details["面积"]
    house["toward"]=details["朝向"]
    return house,response.status_code
```

在上面的代码中,使用 dataCleaning()函数对一条房屋的信息进行数据清洗,以获取房屋的楼层、电梯、面积和朝向等数据。数据清洗包括两部分:删除数据中的空格'\xa0';将数据由列表类型转换成字典类型,以方便存储。例如,一条房屋信息原本的类型是列表类型:['基本信息','面积:89.00㎡','朝向:南','\xa0','维护:5 天前','入住:随时入住','\xa0','楼层:中楼层/28 层','电梯:有','\xa0','车位:暂无数据','用水:暂无数据','\xa0','用电:暂无数据','燃气:有','\xa0','采暖:集中供暖'],经过 dataCleaning()函数后,该房屋类型被转换为字典类型:{'面积':'89.00㎡','朝向':'南','维护':'5 天前','入住':'随时入住','楼层':'中楼层/28 层','电梯':'有','车位':'暂无数据','用水':'暂无数据','用电':'暂无数据','燃气':'有','采暖':'集中供暖'}。dataCleaning()函数的具体实现如下。

```
def dataCleaning(details):
    """
    对房屋信息进行清洗
    :param details: 列表类型,一条房屋信息
    :return: 字典类型,清洗后的房屋信息
    """[1:]
```

```
        details=details
        new_details=list()
        for detail in details:
            if detail== "\xa0":
                continue
            detail=str(detail).split(': ')
            new_details.append(detail)
        return dict(new_details)
```

爬取网页时，不可避免会遇到“\xa0”字符串。“\xa0”其实表示空格。“\xa0”属于latin1(ISO/IEC_8859-1) 中的扩展字符集字符，代表空白符 nbsp(non-breaking space)。latin1 字符集可向下兼容 ASCII 码。

(5) 保存数据。通过 Python 的内置库 CSV 实现对字典数据按行保存。该部分代码如下。


```
def save(row, fileName):
    """
    按行保存数据
    :param row: 字典类型,每一行的数据
    :param fileName: 数据保存的文件名
    """
    with open(fileName, "a+", newline='', encoding='gbk') as f:
        writer=csv.DictWriter(f, fieldnames=fieldnames)
        writer.writerow(row)
```

(6) 主程序。在主函数中，从键盘输入链家网的城市简称和爬取的页码范围，并新建CSV 文件，以保存爬取的数据。然后调用上述步骤中的各个函数进行数据爬取。

所有数据爬取完成后，通过第三方库 Pandas 对数据去除重复行，并在数据中添加一列“ID”作为索引列。至此，爬取任务完成。

该部分代码如下。

```
if __name__ == '__main__':
    city = input("请输入要爬取的城市拼音(如北京: bj,上海: sh): ").strip().lower()
    pageRange=input("请输入要爬取的页码范围(如第 1 页到第 100 页: 1-100): ").strip()
    startPage=int(pageRange.split("-")[0])
    endPage=int(pageRange.split("-")[1]) + 1
    #将爬取的数据保存在 CSV 表
    fileName=city + "_lianJia.csv"
    fieldnames=['floor', 'lift', 'district', 'street', 'community', 'area',
    'toward', 'model', 'rent']
    with open(fileName, "w", newline='') as f:
        #将表头写入 CSV 表
        writer=csv.DictWriter(f, fieldnames=fieldnames)
        writer.writeheader()
    startTime=time.time()
    for page in range(startPage, endPage):
        print("\n--->>正在爬取第" + str(page) + "页")
        houses=getPageLines(city=city, page=page)
        for house in houses:
```